

Windows Workflow Foundation

Hands-On Lab

Lab Manual

Lab 10 – Embedding the Workflow Designer in C#

Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Microsoft Corporation.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user.  Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document.  Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2005 Microsoft Corporation.  All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Visual C# and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**Contents**

[Lab 12: Embedding the Workflow Designer 1](#_Toc134262947)

[Lab Objective 1](#_Toc134262948)

[Exercise 1 – Displaying a Workflow in a Windows Forms Application 1](#_Toc134262949)

[Task 1 – Creating a new Windows Forms Project 1](#_Toc134262950)

[Exercise 2 – Interacting Programmatically with the Workflow Designer in a Windows Forms Application 8](#_Toc134262951)

[Task 1 – Implement Zoom Functionality to the WorkflowDesignerControl 8](#_Toc134262952)

[Task 2 – Design the Zoom functionality Interface 8](#_Toc134262953)

[Exercise 3 – Adding Activities to the Workflow Designer in a Windows Forms Application 12](#_Toc134262954)

[Task 1 – Add an Activity ToolBox to the Workflow Designer 12](#_Toc134262955)

[Task 2 – Enabling the User to Select an Activity in the Workflow Designer 14](#_Toc134262956)

[Task 3 – Enabling the User to Delete an Activity in the Workflow Designer 17](#_Toc134262957)

[Exercise 4 – Programmatically Opening, Saving, Compiling and Running Workflows in the Workflow Designer 20](#_Toc134262958)

[Task 1 – Design the User Interface 20](#_Toc134262959)

[Task 2 – Add Common File Operations 21](#_Toc134262960)

[Task 3 – Bind Activities In the Workflow Designer 28](#_Toc134262961)

[Lab Summary 33](#_Toc134262962)

# Overview

Estimated time to complete this lab: **50 minutes**

The project files for this lab are in the *C:\WF\WF 3.5 Labs\Lab10* folder.

## Objectives

Windows Workflow Foundation provides workflow designer controls for building custom workflow, activity, and rule designers. These are provided so that independent software developers can make better use of Windows Workflow Foundation in line of business applications. The objective of this lab is to demonstrate how to build a simple Windows Forms workflow designer using the workflow designer control. After completing this lab, you will be able to:

* Display a Workflow in a Windows Forms application
* Interact programmatically with the workflow designer control in a Windows Forms application
* Programmatically add activities to the workflow designer control in a Windows Forms application
* Programmatically open, save, compile and run workflows

This lab has a lot of code which needs to be added to the projects. The soft copy of this lab document is available to you on the lab machine. It is recommended that you use the provided code snippets as you work through the lab.

More information about Windows Workflow Foundation can be found at <http://msdn.microsoft.com/workflow>

## System Requirements

* Microsoft Visual Studio 2008

## Setup

Unzip the lab to your local hard drive. Everything needed to complete the lab is in the zip file.

### Physical Folder Structure

File paths referenced in this lab assume the lab is installed in the following folder:

*C:\WF\WF 3.5 Labs\Lab[Number]*

Within the ***Lab[Number]*** folder, several child folders are available:

* ***CSharp*** – The lab written for C#
* ***VB*** – The lab written for VB
* ***resources*** –Any files referenced in the lab can be found in the Resources subdirectory, including source code for custom assemblies referenced in the exercises.

Within each *[Language]* folder, several child folders are available:

* ***before*** – The work area for completing the HOL
* ***after*** – The fully completed HOL

### Code Snippets

All code required for this lab consisting of more than 2 lines is available as code snippets. To learn more about code snippets including how to install them and how to use them, see the snippet guide document for the language of your choice in the folder:

*C:\WF\WF 3.5 Labs\Snippets*

## Starting Material

### Acronyms Used in this Lab

* WF – Windows Workflow Foundation

### Scenario

# Exercise 1 – Displaying a Workflow in a Windows Forms Application

In this exercise, you will create a Windows Forms application that will be comprised of a **ToolStrip** containing simple Windows controls such as **Buttons** and a Workflow design surface, which will initially display an empty workflow.

## Task 1 – Creating a new Windows Forms Project

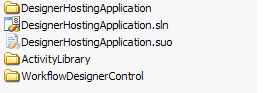
1. Open Microsoft Visual Studio 2005, click **Start | Programs | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005**.
2. In Visual Studio 2005, click **File | New | Project.**
3. In the New Project dialog box, in the Project types tree on the left expand **Visual C# | Windows**.
4. Select the **Windows Application** template and enter the following values for the **Name** and **Location**:

**Name:** *DesignerHostingApplication*

**Location:** *C:\WF\WF 3.5 Labs\Lab10*

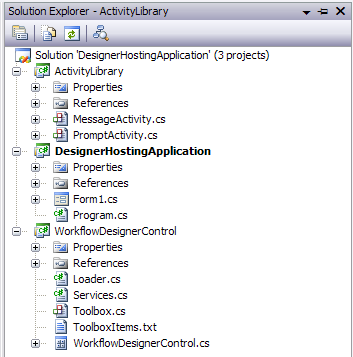
1. Before we begin creating our designer host application we must add two projects to our solution.
   * **WorkflowDesignerControl**: This project contains our workflow display control
   * **ActivityLibrary**: This project contains two custom Workflow activities which we will use throughout the lab.
2. Copy the following two project directories, *WorkflowDesignerControl* and *ActivityLibrary* from the *C:\WF\WF 3.5 Labs\Lab10\Resources* to *C:\WF\WF 3.5 Labs\Lab10\DesignerHostingApplication*.

Your working directory should now look similar to this:

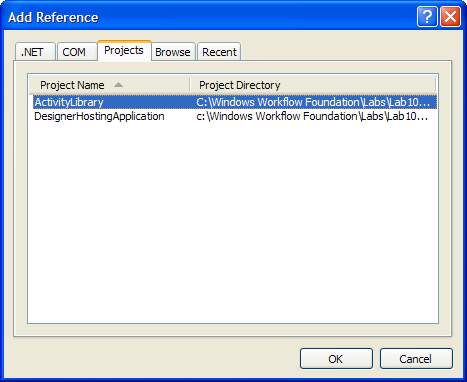


1. To add the projects to our solution, right-click the Solution file in the **Solution Explorer** and click **Add Existing Project.**
2. In the **Open File Dialog**, navigate to *C:\WF\WF 3.5 Labs\Lab10\* *DesignerHostingApplication\WorkflowDesignerControl* and select the *WorkflowDesignerControl.csproj* file.
3. Right-click the solution file a second time and select **Add Existing Project**.
4. In the Open File Dialog, navigate to *C:\WF\WF 3.5 Labs\Lab10\* *DesignerHostingApplication\ ActivityLibrary* and select the **ActivityLibrary.csproj** file.

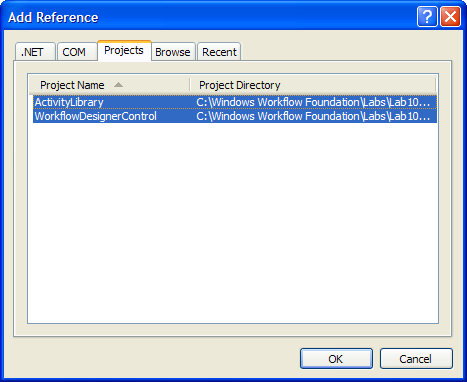
Your solution should now look similar to this



1. In the **Solution Explorer**, right-click the *WorkflowDesignerControl* project and select **Add Reference**.
2. In the **Add Reference** window, select the **Projects** tab and select *ActivityLibrary*.



1. Click **OK**.
2. In the **Solution Explorer**, right-click the *DesignerHostingApplication* project and select **Add Reference**.
3. In the **Add Reference** window, click the **Projects** tab and select *ActivityLibrary* and *WorkflowDesignerControl*.



1. Click **OK**.
2. In the **Solution Explorer**, right-click the *DesignerHostingApplication* project and select **Add Reference**.
3. In the **Add Reference** window, click the **.Net** tab and select the following components:

* System.Design
* System.Drawing.Design
* System.Workflow.Activities
* System.Workflow.ComponentModel

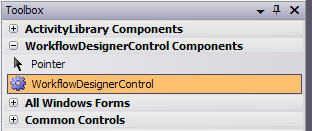
1. In the **File** menu select **Save All** or press **CTRL + SHIFT + S**,
2. In the **Build** menu select **Build Solution** or press **F6**, ensure all three projects build with no errors.

## Task 2 – Defining the User Interface for the Windows Application

1. In **Solution Explorer**, right-click *Form1.cs*, and click **Rename**. Rename *Form1.cs* to *DesignerShell.cs*. Accept the dialog to rename all references.
2. In **Solution Explorer**, right-click *DesignerShell.cs,* and click **View Designer**.
3. Click the form in the designer and press **F4** to open the **Properties** window. Set the **Text** property of the form to *DesignerShell*.
4. Set the **Size** property to **Width**: *600* and **Height**: *500*.
5. Open the **Toolbox** if it is not already open, and expand the **All Windows Forms** node in the **Toolbox**.
6. Drag and drop a **Toolstrip** onto the design surface of the form and set the properties as follows:
   * **Name**: *toolStrip*
   * **GripStyle**: *Hidden*
   * **RenderMode**: *System*
   * Ensure the **Dock** property is set to *Top*

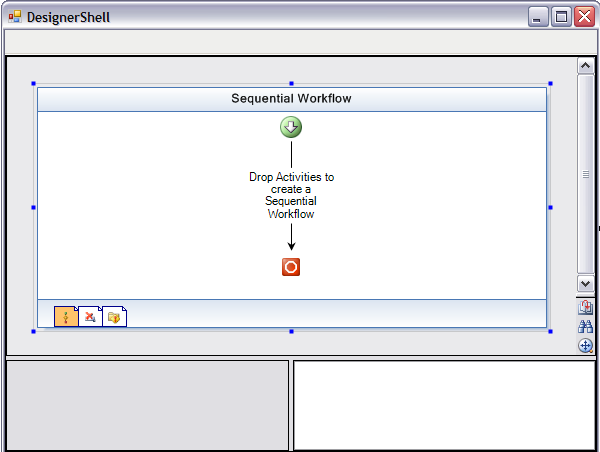
**

1. Return to the **Toolbox** and expand the *WorkflowDesignerControl* Components node.
2. You should see a *WorkflowDesignControl* which has been built from our *WorflowDesignControlProject*. Drag and drop a *WorkflowDesignControl* onto the design surface of the form just under the **toolStrip**.



1. Set the properties of the control as follows:
   * **Name**: *workflowDesignerControl*
   * **Dock**: *Fill*

* Your form should look similar to this:



1. Press **F6** to build the solution, verify that all the projects build successfully

# Exercise 2 – Interacting Programmatically with the Workflow Designer in a Windows Forms Application

In this exercise, you will write code that responds to events and that can manipulate activities in the embedded Workflow Designer. You will also write code that enables the user to zoom in the Workflow Designer.

## Task 1 – Implement Zoom Functionality to the WorkflowDesignerControl

1. In the **Solution Explorer**, right-click on *WorkflowDesignerControl.cs* and select **View Code**.
2. Insert the following method at the end of *WorkflowDesignerControl* class just after the *LoadWorkFlow* method, to support zooming:

(Snippet: WFLab10\_Ex02\_Task01\_ProcessZoom)

**public void ProcessZoom(int zoomFactor)**

**{**

**this.workflowView.Zoom = zoomFactor;**

**this.workflowView.Update();**

**}**

1. Rebuild the solution by pressing **F6**.

## Task 2 – Design the Zoom functionality Interface

In this task we will build the user interface to allow the user to zoom the workflow designer in and out using three predefined zoom levels 25%, 100% and 200%.

1. In the **Solution Explorer**, right-click on *DesignerShell.cs* and select **View Code**.
2. Insert the following method to invoke the zoom functionality in the *DesignerShell*class just below the constructor.

(Snippet: WFLab10\_Ex02\_Task02\_ZoomMenuItem\_Click)

private void zoomDropDownMenuItem\_Click(object sender, EventArgs e)

{

if (sender is ToolStripMenuItem)

{

ToolStripMenuItem menuItem = (ToolStripMenuItem)sender;

int zoomFactor = 0;

bool result = Int32.TryParse(menuItem.Tag.ToString(), out zoomFactor);

if (result)

{

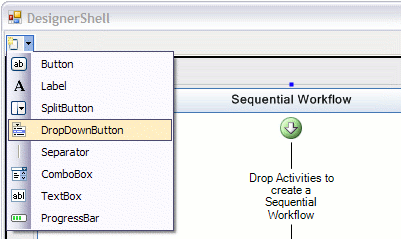
this.workflowDesignerControl.ProcessZoom(zoomFactor);

}

}

}

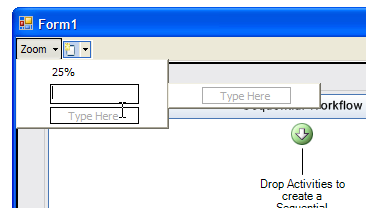
1. In the **Solution Explorer**, right-click on *DesignerShell.cs* and select **View Designer**.
2. Select the **toolStrip** control at the top of the form.
3. Using the **SmartTasks** tool, add a **DropDownButton** to the **toolStrip**.



1. The new **DropDownButton** will be created, select it and set it’s properties as follows:

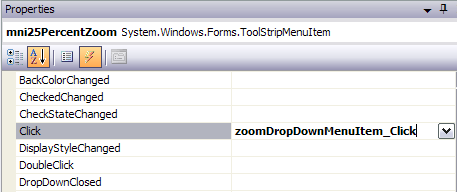
* **Name**: *zoomDropDown*
* **DisplayStyle**: *Text*
* **Text**: *Zoom*

1. Click the zoomDropDown Button and using the designer add three items to the drop down.
   * Item01
     1. **Text**: *25%*
   * Item02
     1. **Text**: *100%*
   * Item03
     1. **Text**: *200%*



Now that we have created our drop down options we need to set the someparameters and event handlers for each. We will use the **Tag** to store the actual zoom factor for the each item.

1. Click on the first item in the list *(25%)* and press **F4** to view the properties window. Set the properties as follows:
   * **Name**: *mni25PercentZoom*
   * **Tag**: *25*
2. Click on the events icon at the top of the properties window Screen 00012
3. Set the **Click** event handler to *zoomDropDownMenuItem\_Click* using the drop down options



1. Click on the second item in the list (*100%*) and set the properties as follows:
   * **Name**: *mni100PercentZoom*
   * **Tag**: *100*
2. Click on the events icon at the top of the properties window Screen 00012
3. Set the **Click** event handler to *zoomDropDownMenuItem\_Click* using the drop down options
4. Click on the last item in the list (*200%*) and set the properties as follows:
   * **Name**: *mni200PercentZoom*
   * **Tag**: *200*
5. Click on the events icon at the top of the properties window Screen 00012
6. Set the **Click** event handler to *zoomDropDownMenuItem\_Click* using the drop down options
7. Click **Debug | Start Debugging** or press **F5** to run the project. The application window should appear.
8. Click the *Zoom* drop down button in the Tool Strip and select *25%.* Verify that the workflow in the Workflow Designer is displayed **quarter** its original size.
9. Click the *Zoom* drop down button in the Tool Strip and select *200%.* Verify that the workflow in the Workflow Designer is displayed **double** its original size.
10. Close the application window.

# Exercise 3 – Adding Activities to the Workflow Designer in a Windows Forms Application

In this exercise, you will write code to add Activities to a Workflow using a toolbox style control. You will also write code that enables the user to select an activity and change its properties, and to delete an activity from the workflow.

## Task 1 – Add an Activity ToolBox to the Workflow Designer

1. Start with the solution as created in Exercise 2.
2. In **Solution Explorer**, right-click *WorkflowDesignerControl.cs*, and click **View Code**.
3. Insert the following code into the *WorkflowDesignerControl* class constructor, directly after the **InitializeComponent** method call, to add a custom Toolbox control to the *WorkflowDesignerControl*.

(Snippet: WFLab10\_Ex03\_Task01\_ToolBox)

Toolbox toolbox = new Toolbox(this);

this.propertyGridSplitter.Panel1.Controls.Add(toolbox);

toolbox.Dock = DockStyle.Fill;

toolbox.BackColor = BackColor;

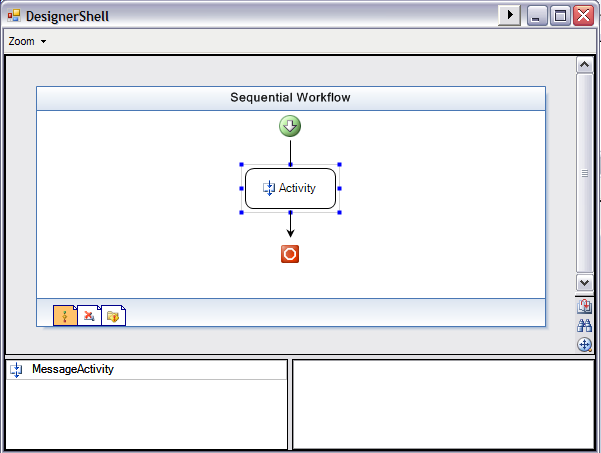
toolbox.Font = WorkflowTheme.CurrentTheme.AmbientTheme.Font;

1. Now we want to include our custom *MessageActivity* to the Activity ToolBox so we can use it in our Workflow. *MessageActivity* is a simple activity that has a *Message* string property. When executed, the message is displayed in a **MessageBox**.
2. Add the following line of text to the *ToolBoxItems.txt* file found in the *WorkflowDesignControl* project.

ActivityLibrary.MessageActivity, ActivityLibrary

**Note:** At runtime the Toolbox control will read the types specified in this file and load them into the Toolbox.

1. Save all files by pressing **CTRL + SHIFT + S** and run the solution by pressing **F5**
2. The Activity ToolBox should appear in the bottom left hand panel and should include a single Activity named *MessageActivity*.
3. Test the application by dragging and dropping a *MessageActivity* from the Activity ToolBox onto the Workflow Designer.



1. Exit the *DesignerShell* program.

## Task 2 – Enabling the User to Select an Activity in the Workflow Designer

Now that we can add activities to the workflow, we need to allow the user to select activities and edit their properties.

1. In **Solution Explorer**, right-click *WorkflowDesignerControl.cs*, and click **View Code**.
2. Locate the *LoadWorkflow(string xoml)* method in the *WorkflowDesignerControl* class and add the following code highlighted in grey just after line *this.workflowView.Focus().*

(Snippet: WFLab10\_Ex03\_Task02\_SelectionChanged)

private void LoadWorkflow(string xoml)

{

SuspendLayout();

DesignSurface designSurface = new DesignSurface();

WorkflowLoader loader = new WorkflowLoader();

loader.Xoml = xoml;

designSurface.BeginLoad(loader);

IDesignerHost designerHost = designSurface.GetService(typeof(IDesignerHost)) as IDesignerHost;

if (designerHost != null && designerHost.RootComponent != null)

{

IRootDesigner rootDesigner = designerHost.GetDesigner(designerHost.RootComponent) as IRootDesigner;

if (rootDesigner != null)

{

UnloadWorkflow();

this.designSurface = designSurface;

this.loader = loader;

this.workflowView = rootDesigner.GetView(ViewTechnology.Default) as WorkflowView;

this.workflowViewSplitter.Panel1.Controls.Add(this.workflowView);

this.workflowView.Dock = DockStyle.Fill;

this.workflowView.TabIndex = 1;

this.workflowView.TabStop = true;

this.workflowView.HScrollBar.TabStop = false;

this.workflowView.VScrollBar.TabStop = false;

this.workflowView.Focus();

ISelectionService selectionService = GetService(typeof(ISelectionService)) as ISelectionService;

if (selectionService != null)

{

selectionService.SelectionChanged += new EventHandler(OnSelectionChanged);

}

}

}

ResumeLayout(true);

}

1. And add the following event handler to the WorkflowDesignerControl class just below the *ProcessZoom* method we added earlier.

(Snippet: WFLab10\_Ex03\_Task02\_OnSelectionChanged)

private void OnSelectionChanged(object sender, EventArgs e)

{

ISelectionService selectionService = GetService(typeof(ISelectionService)) as ISelectionService;

if (selectionService != null)

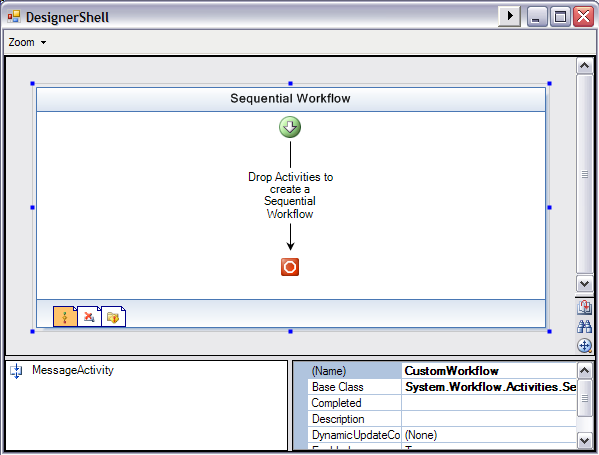
{

this.propertyGrid.SelectedObjects = new ArrayList(selectionService.GetSelectedComponents()).ToArray();

}

}

1. Now when the selected activity changes within the *WorkflowDesignerControl*, the **PropertyGrid** control located in the bottom right hand corner will load the properties for the selected activity.
2. Press **F5** to run the application and test the activity selection.
3. Your form should now look similar to this:



1. Drag and drop two *MessageActivities* onto the Workflow.
2. Select the first *MessageActivity* and set it’s properties as follows
   * **Name**: *activity01*
   * **Description**: *“This is message activity 01”*
3. Select the second *MessageActivity* and set it’s properties as follows
   * **Name**: *activity02*
   * **Description**: *“This is message activity 02”*
4. You will notice that as you modify each activities properties the changes are reflected in the designer.
5. Close the application.

## Task 3 – Enabling the User to Delete an Activity in the Workflow Designer

1. In **Solution Explorer**, right-click *WorkflowDesignerControl.cs*, and click **View Code**.
2. Insert the following method into the *WorkflowDesignerControl* class just below the *OnSelectionChanged* method to add functionality to delete the selected Activity.

(Snippet: WFLab10\_Ex03\_Task03\_DeleteSelected)

public void DeleteSelected()

{

ISelectionService selectionService = (ISelectionService) this.GetService(typeof(ISelectionService));

if (selectionService != null)

{

if (selectionService.PrimarySelection is Activity)

{

Activity activity = (Activity)selectionService.PrimarySelection;

if (activity.Name != this.WorkflowName)

{

activity.Parent.Activities.Remove(activity);

this.workflowView.Update();

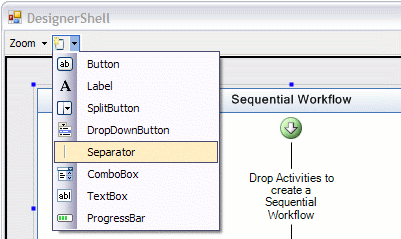
}

}

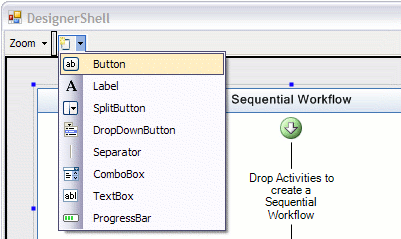
}

}

1. Press **F6** to build the solution.
2. In the **Solution Explorer** right-click *DesignerShell.cs* and select **View Designer**.
3. Select the *toolStrip* control and using the **Smart Task** tool add a **Seperator**.



1. Using the **Smart Task** tool again add a new **Button**.



1. Set the buttons properties as follows:
   * **Name**: *btnDelete*
   * **DisplayStyle**: *Text*
   * **Text**: *“Remove Activity”*
2. Now return to the designer and double click *btnDelete* to create the default Click handler.
3. Within the newly created event handler add the following line of code:

private void btnDelete\_Click(object sender, EventArgs e)

{

this.workflowDesignerControl.DeleteSelected();

}

1. Press **F5** to build and run the application.
2. Drag and drop two or more *MessageActivities* onto the workflow.
3. Select one of the *MessageActivities* within the workflow and click the *Remove Activity* button.
4. Verify that the activity is removed from the workflow.
5. Close the application.

# Exercise 4 – Programmatically Opening, Saving, Compiling and Running Workflows in the Workflow Designer

## Task 1 – Design the User Interface

1. Start with the solution as created in Exercise 3.
2. In **Solution Explorer**, double-click *DesignerShell.cs* to show the designer view.
3. Select the *toolStrip* control and use the **Smart Task** tool to add a **Seperator**.
4. Use the **Smart Task** tool again and add a new **Button**.
5. Set the **Button** properties as follows:
   * **Name**: *btnOpen*
   * **DisplayStyle**: *Text*
   * **Text**: *Open*
6. Your form should now look similar to the following:

Screen 00000

1. Using the **Smart Task** tool, add three more **Buttons** to the *toolStrip* and set their properties as follows:
   * Button01
     1. **Name**: *btnSave*
     2. **DisplayStyle**: *Text*
     3. **Text**: “*Save”*
   * Button02
     1. **Name**: *btnCompile*
     2. **DisplayStyle**: *Text*
     3. **Text**: ‘*Compile”*
   * Button03
     1. **Name**: *btnRun*
     2. **DisplayStyle**: *Text*
     3. **Text**: “*Run”*
2. Your form should look similar to the following:

Screen 00001

## Task 2 – Add Common File Operations

1. In **Solution Explorer**, right-click *WorkflowDesignerControl* and select **View Code**.
2. Add the following method to the *WorkflowDesignerControl* class just after the *DeleteSelected* method to add functionality to open existing workflows from file.

(Snippet: WFLab10\_Ex04\_Task02\_LoadExistingWorkflow)

public void LoadExistingWorkflow()

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "xoml files (\*.xoml)|\*.xoml|All files (\*.\*)|\*.\*";

openFileDialog.FilterIndex = 1;

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

using (XmlReader xmlReader = XmlReader.Create(openFileDialog.FileName))

{

WorkflowMarkupSerializer serializer = new WorkflowMarkupSerializer();

this.workflow = (SequentialWorkflowActivity)serializer.Deserialize(xmlReader);

this.LoadWorkflow();

this.XomlFile = openFileDialog.FileName;

this.Text = "Designer Hosting Sample -- [" + openFileDialog.FileName + "]";

}

}

}

1. Add the following methods to the *WorkflowDesignerControl* class just after the *LoadExistingWorkflow* method to add functionality to save workflows to file.

(Snippet: WFLab10\_Ex04\_Task02\_Save)

private void SaveFile()

{

if (this.XomlFile.Length != 0)

{

this.SaveExistingWorkflow(this.XomlFile);

}

else

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "xoml files (\*.xoml)|\*.xoml|All files (\*.\*)|\*.\*";

saveFileDialog.FilterIndex = 1;

saveFileDialog.RestoreDirectory = true;

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

this.SaveExistingWorkflow(saveFileDialog.FileName);

this.Text = "Designer Hosting Sample -- [" + saveFileDialog.FileName + "]";

}

}

}

internal void SaveExistingWorkflow(string filePath)

{

if (this.designSurface != null && this.loader != null)

{

this.XomlFile = filePath;

this.loader.PerformFlush();

}

}

public bool Save()

{

return this.Save(true);

}

public bool Save(bool showMessage)

{

Cursor cursor = this.Cursor;

this.Cursor = Cursors.WaitCursor;

bool saveOK = true;

try

{

// Save the workflow first, and capture the filePath of the workflow

this.SaveFile();

XmlDocument doc = new XmlDocument();

doc.Load(this.XomlFile);

XmlAttribute attrib = doc.CreateAttribute("x", "Class", "http://schemas.microsoft.com/winfx/2006/xaml");

attrib.Value = string.Format("{0}.{1}", this.GetType().Namespace, this.WorkflowName);

doc.DocumentElement.Attributes.Append(attrib);

doc.Save(this.XomlFile);

if (showMessage)

{

MessageBox.Show(this, "Workflow generated successfully. Generated xoml file:\n" + Path.Combine(Path.GetDirectoryName(this.GetType().Assembly.Location), this.XomlFile), this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

saveOK = false;

}

finally

{

this.Cursor = cursor;

}

return saveOK;

}

1. Add the following methods to the *WorkflowDesignerControl* class just after the *Save(bool showMessage)* method to add functionality to compile your workflows.

(Snippet: WFLab10\_Ex04\_Task02\_Compile)

public bool Compile()

{

return this.Compile(true);

}

public bool Compile(bool showMessage)

{

if (!this.Save(false))

{

return false;

}

if (!File.Exists(this.XomlFile))

{

MessageBox.Show(this, "Cannot locate xoml file: " + Path.Combine(Path.GetDirectoryName(this.GetType().Assembly.Location), XomlFile), this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

return false;

}

bool compileOK = true;

Cursor cursor = this.Cursor;

this.Cursor = Cursors.WaitCursor;

try

{

// Compile the workflow

String[] assemblyNames = { AdditionalAssembies };

WorkflowCompiler compiler = new WorkflowCompiler();

WorkflowCompilerParameters parameters = new WorkflowCompilerParameters(assemblyNames);

parameters.LibraryPaths.Add(Path.GetDirectoryName(typeof(ActivityLibrary.MessageActivity).Assembly.Location));

parameters.GenerateInMemory = true;

WorkflowCompilerResults compilerResults = compiler.Compile(parameters, this.XomlFile);

inMemoryAssembly = compilerResults.CompiledAssembly;

StringBuilder errors = new StringBuilder();

foreach (CompilerError compilerError in compilerResults.Errors)

{

errors.Append(compilerError.ToString() + '\n');

}

if (errors.Length != 0)

{

MessageBox.Show(this, errors.ToString(), this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

compileOK = false;

}

else if (showMessage)

{

MessageBox.Show(this, "Workflow compiled successfully. Compiled assembly:\n" + compilerResults.CompiledAssembly.GetName(), this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

finally

{

this.Cursor = cursor;

}

return compileOK;

}

Notice the line above which reads *parameters.GenerateInMemory = true;* This means that we are generating the assembly in memory and not writing to file. If you want to generate a .dll for your workflow you could try the following

*parameters.GenerateInMemory = false;*

*parameters.OutputAssembly = string.Format("{0}.dll", this.WorkflowName);*

1. Add the following methods to the *WorkflowDesignerControl* class just after the *Compile(bool showMessage)* method to add functionality to run your workflow.

(Snippet: WFLab10\_Ex04\_Task02\_Run)

public bool Run()

{

if (inMemoryAssembly == null)

{

if (!this.Compile(false))

{

return false;

}

}

// Start the runtime engine

if (this.workflowRuntime == null)

{

this.workflowRuntime = new WorkflowRuntime();

this.workflowRuntime.StartRuntime();

}

this.workflowRuntime.WorkflowCompleted += new EventHandler<WorkflowCompletedEventArgs>(workflowRuntime\_WorkflowCompleted);

string typeName = string.Format("{0}.{1}", this.GetType().Namespace, this.WorkflowName);

this.workflowRuntime.CreateWorkflow(AppDomain.CurrentDomain.CreateInstanceAndUnwrap(inMemoryAssembly.FullName, typeName).GetType()).Start();

return true;

}

void workflowRuntime\_WorkflowCompleted(object sender, WorkflowCompletedEventArgs e)

{

MessageBox.Show("Workflow complete");

}

1. In the **Solution Explorer** right-click the *DesignerShell.cs* file and selct **View Code**.
2. Insert the following event handlers into the *DesignerShell* class just after the *btnDelete\_Click* event handler

(Snippet: WFLab10\_Ex04\_Task02\_EventHandlers)

private void btnSave\_Click(object sender, EventArgs e)

{

this.workflowDesignerControl.Save();

}

private void btnCompile\_Click(object sender, EventArgs e)

{

this.workflowDesignerControl.Compile();

}

private void btnRun\_Click(object sender, EventArgs e)

{

this.workflowDesignerControl.Run();

}

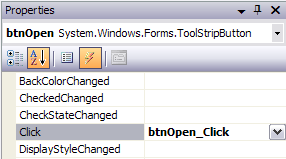
private void btnOpen\_Click(object sender, EventArgs e)

{

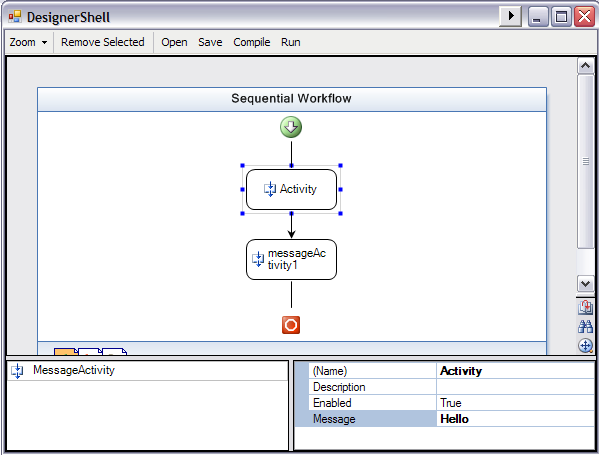
this.workflowDesignerControl.LoadExistingWorkflow();

}

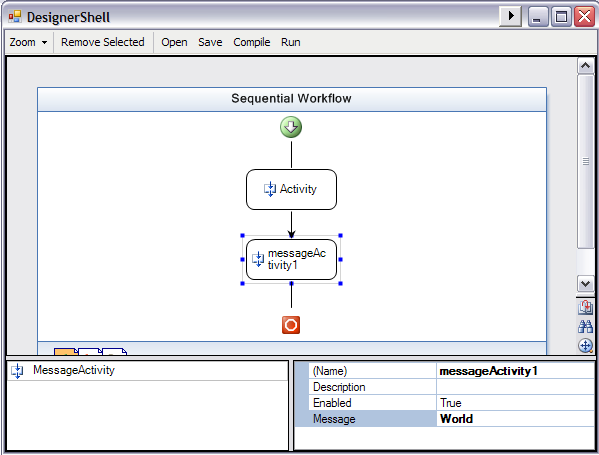
1. Return to the form designer for the *DesignerShell* form.
2. Select *btnOpen* and press **F4** to open the properties window,
3. Click on the events icon at the top of the properties window Screen 00012
4. Set the Click event handler to *btnOpen\_Click* using the drop down options



1. With the property window still open, select *btnSave* on the form designer.
2. Set the Click event handler to *btnSave\_Click* using the drop down options
3. Using the same process set the click handler for *btnCompile* to *btnCompile\_Click* and the click handler for *btnRun* to *btnRun\_Click*.
4. Press **F5** to run the application.
5. To verify our application is running correctly drag and drop two *MessageActivities* onto the Workflow.
6. Select the first *MessageActivity* and set its message property to *“Hello”.*



1. Select the second *MessageActivity* and set its message property to *“World”.*



1. Click the **Run** button, this will save, compile and run the workflow. As we have not yet saved our workflow you will be prompted to save before the application can compile and run your workflow. In the **Save File** Dialog save your workflow to *C:\WF\WF 3.5 Labs\Lab10\DesignerHostingApplication\CustomWorkflow.xoml* and click **Save***.* Once you have saved the application will continue to compile and run your workflow.
2. Verify that you see three message boxes appear, the first saying “*Hello”,* the second saying *“World”* and the third informing you that the Workflow has completed successfully.

## Task 3 – Bind Activities in the Workflow Designer

For this task we are going to bind to activities within our workflow. The first activity will display a question for the user and allow them to enter an answer, the second activity will be bound to the first activity and will display the users answer in a **MessageBox**.

We will add a second custom activity to the Activity Toolbox called *PromptActivity***.** *PromptActivity* is a simple activity that provides the functionality to ask the user our question and receive the answer.

1. To add the *PromptActivity* to our Activity Toolbox add the following gray highlighted line of text to the *ToolboxItems.txt* file found in the *WorkflowDesignerControl* project.

ActivityLibrary.MessageActivity, ActivityLibrary

ActivityLibrary.PromptActivity, ActivityLibrary

1. Now, we want to be able to bind the *Message* property of a *MessageActivity* to a *PromptActivity’s* *Answer* property, to do this we must modify the *MessageActivity* slightly. Right-click on *MessageActivity.cs*in the *ActivityLibrary* project and select **View Code**.
2. Delete the private *message* field and public *Message* property from the *MessageActivity* class

private string message;

public string Message

{

get { return message; }

set { message = value; }

}

1. And insert the following lines of code in it’s place

(Snippet: WFLab10\_Ex04\_Task03\_MessageProperty)

public static DependencyProperty MessageProperty = System.Workflow.ComponentModel.DependencyProperty.Register("Message", typeof(string), typeof(MessageActivity));

[Description("The Message to display in the MessageBox")]

[Browsable(true)]

[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]

public string Message

{

get

{

return ((string)(base.GetValue(MessageActivity.MessageProperty)));

}

set

{

base.SetValue(MessageActivity.MessageProperty, value);

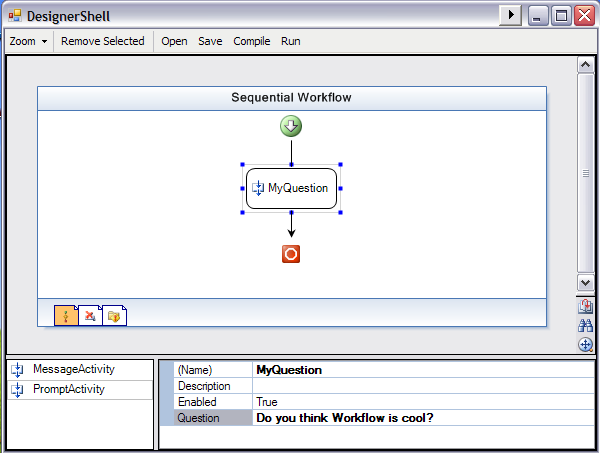
}

}

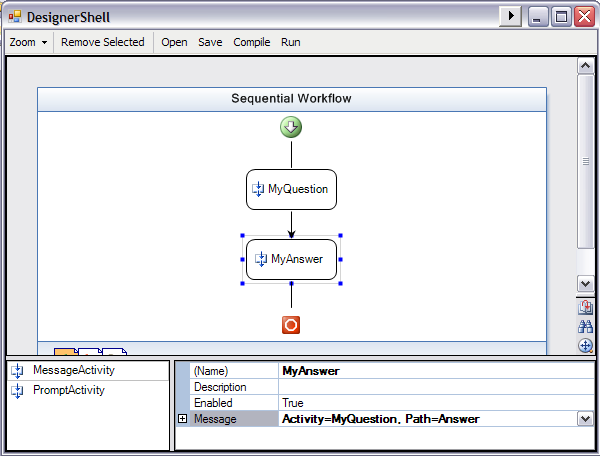
1. Locate the **Execute** method in the *MessageActivity* class and Capitalize ‘*m*’ in *this.message*, to reflect the changes we made to the *Message* property

MessageBox.Show(this.Message);

1. Run the application by pressing **F5**.
2. Notice our new *PromptActivity* appears in the Activity Toolbox.
3. Drag and drop a *PromptActivity* onto the workflow and set it’s properties as follows:
   * **Name**: *MyQuestion*
   * **Question**: *“Do you think Workflow is cool?”*

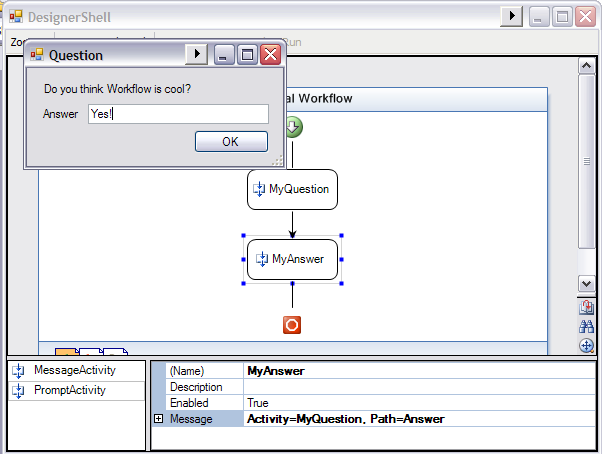


1. Drag and drop a *MessageActivity* onto the workflow after your *MyQuestion* activity and set it’s properties as follows:
   * **Name**: *MyAnswer*
   * **Message**: *Activity=MyQuestion, Path=Answer*



Notice the syntax we have entered for the *Message* property. What we are saying is: *For the* ***Message*** *property get your value from the activity named* ***MyQuestion*** *and the property named* ***Answer***

1. Click the *Run* Button
2. When prompted, save the workflow by overwriting *CustomWorkflow.xoml*
3. When the question form appears type *“Yes!”* in the answer box.



1. Click **OK**
2. Another **MessageBox** should appear with the text *“Yes!”.*

You have successfully bound the *MyAnswer* activity to *MyQuestion*.

1. To complete this lab add the following gray highlighted text into the *ToolboxItems.txt* file to include some other common workflow activities

ActivityLibrary.MessageActivity, ActivityLibrary

ActivityLibrary.PromptActivity, ActivityLibrary

System.Workflow.Activities.DelayActivity

System.Workflow.Activities.HandleExternalEventActivity

System.Workflow.Activities.IfElseActivity

System.Workflow.Activities.CallExternalMethodActivity

System.Workflow.Activities.InvokeWebServiceActivity

System.Workflow.Activities.InvokeWorkflowActivity

System.Workflow.Activities.ListenActivity

System.Workflow.Activities.ParallelActivity

System.Workflow.Activities.ReplicatorActivity

System.Workflow.Activities.SequenceActivity

System.Workflow.Activities.WhileActivity

System.Workflow.Activities.EventHandlingScopeActivity

1. Press **F5** to run the application
2. Verify that the Activity Toolbox now contains our two custom activities and some common built-in Workflow Activities.
3. Close the application.

# Lab Summary

In this lab you performed the following exercises.

* Display a Workflow in a Windows Forms application
* Interact programmatically with the Workflow Designer in a Windows Forms application
* Programmatically add activities to the Workflow Designer in a Windows Forms application
* Programmatically open, save, compile and run workflows in the Workflow Designer

In this lab, you created a Windows Forms application that has an embedded Workflow Designer. You wrote code to display a sequential workflow, and added support for zooming operations. You also wrote code to enable the user to add, select, and delete Code Activities in the Workflow Designer. Finally, you wrote code to enable the user to open, save, compile and run workflows and demonstrated activity binding